A Machine Learning Approach Using Classifier Cascades for Optimal Routing in Opportunistic Internet of Things Networks

Vidushi Vashishth*, Anshuman Chhabra†, Deepak Kumar Sharma*

*Division of Information Technology
Netaji Subhas Institute of Technology, University of Delhi
New Delhi, India
Email: vidushivashishth96@gmail.com, dk.sharma1982@yahoo.com

†Department of Computer Science
University of California
Davis, USA
Email: chhabra@ucdavis.edu

***Abstract*—Routing in Opportunistic Internet of Things Network (OppIoT) is an involved problem, because the network is intermittently connected and source to destination end-to-end paths are non-existent. Moreover, Machine Learning (ML) has recently achieved great success in multiple domains and is now being applied to automate routing in Opportunistic Networks (OppNets) which are similar in characteristics to OppIoT, through protocols such as MLProph and KNNR. In this paper, we utilize cascade learning, a form of ensemble based ML, for improved routing in OppIoT. Through simulations we show that our proposed protocol called Cascaded Machine Learning based routing protocol (CAML), outperforms existing ML based protocols (MLProph and KNNR), and traditional well-performing protocols (HBPR and PRoPHET), on a wide range of performance metrics including message delivery probability, average hop count, packets dropped and network overhead ratio.**

***Index Terms*—Internet of Things, Opportunistic Networks, Routing Protocols, Machine Learning, Neural Networks, Cascade Learning, Logistic Regression, ONE Simulator, Delay Tolerant Networks**

I. Introduction

OPPORTUNISTIC Internet of Things Networks (OppIoT) are a sub-class of Delay Tolerant Networks (DTNs) that find applications in areas of sparse network connectivity. They were introduced in [1] to overcome the geographical limitations of IoT applications due to limited network infrastructure availability. OppIoTs are different form traditional IoT networks as end-to-end routing paths between the source and destination do not exist in OppIoT. Devices hence incorporate the store-carry-forward principle for message transmission. Here, *relay* nodes store messages in a buffer and *opportunistically* forward them to other nodes that come in contact with them, so as to get these messages closer to their destinations. Thus, the very characteristics of OppIoT (intermittent network connectivity and a lack of fixed infrastructure), make designing a routing protocol capable of ensuring a high rate of successful message delivery, a challenging problem to solve. OppIoT share this message-forwarding behavior with another sub-class of DTNs called Opportunistic Networks (OppNet). This enables the same routing designs to be applicable to both OppNets and OppIoTs. Earlier works such as PRoPHET [2] and HBPR [3] have used the concept of node encounters and history to improve routing decisions in an OppNet. PRoPHET does this by a delivery predictability metric which encourages nodes that frequently come in contact with each other to relay messages to each other. HBPR exploits the patterns of node's movements and encounter history to predict the node's next location and aid message transmission.

Recently, ML has also been used in automating routing decisions in OppNets through the MLProph [4] and KNNR [5] routing protocols. However, these protocols directly use ML algorithms such as Neural Networks, Decision Trees and K-Nearest Neighbor for automated routing but do not analyze the data which the classifier is training on. Statistical analysis of the data is imperative for ascertaining the number and type(s) of classifiers that should be used to obtain optimal results. We present such a data analysis in this paper to create an optimal routing protocol for OppIoT. We dive into the details of this motivation in Section III. We justify our choice of using a two-stage cascade model with a Logistic Regression [6] classifier as the first stage coupled with a Multi-Layer Perceptron (MLP) Neural Network classifier as the second stage.

Cascade learning is an ensemble based ML technique that is generally used in cases where the dataset possesses a class imbalance problem. In binary classification, if the number of samples for a particular class outweigh the number of samples present for the other class, employing just a general ML algorithm for classification is not suitable and gives skewed results. Instead, cascades of ML classifiers can be used where each classifier passes on it's knowledge of the dataset as input to the other classifier. This approach has been used in literature to produce state-of-the-art results on a variety of classification problems [7]–[9].

978-1-7281-1207-7/19/$31.00 ©2019 IEEE

The rest of the paper is structured as follows: Section II describes related work in the field, Section III details the motivation for the proposed work, Section IV delineates the working of the proposed routing protocol, Section V covers the details of simulations performed and subsequent results obtained, and Section VI concludes with a summary of the paper's contribution.

II. Related Work

Owing to the similarity in routing behavior of OppNets and OppIoT, we compare the proposed routing protocol for OppIoT in this paper to some state of the art routing protocols for OppNets. Routing techniques for OppNets can be categorized into two major types: *context-aware* and *context-oblivious*. Context-aware routing protocols are protocols that utilize the context information about the nodes in the network for making routing decisions. Then, the problem of choosing the next hop for a message first involves extracting certain information/features from the nodes. These features are then provided as input to an algorithm that makes informed routing decisions. Protocols such as HBPR, PRoPHET, CAR [10], MaxProp [11], MLProph, KNNR and HiBOp [12], among others, are context-aware protocols.

In HBPR, the network is divided into *cells* of equal areas, and context information such as the node's speed, it's home cell, and the direction of it's movement, are used for predicting it's future movements, allowing for improved routing. PRoPHET is a seminal work in the field, and uses information such as it's frequency of interaction with other nodes and a transitive property to compute a probabilistic estimate of whether relaying the message to a particular node will lead to eventual delivery. MaxProp builds upon this idea and also utilizes context information such as the frequency of node's movements with respect to certain physical locations. CAR divides nodes into separate clouds, and inter-cloud communication is enabled by finding nodes in the current cloud which have a strong likelihood of delivering the message. In HiBOp, an Identity table and a History table are used to store relevant information about the node and it's neighbors, which are then used to decide the number of copies of the message to be disseminated along with the next choice of relay.

MLProph and KNNR are context-aware algorithms that utilize ML algorithms for making intelligent routing decisions after observing some features of the nodes. In the MLProph paper, the authors use both MLP Neural Networks and Decision Trees for deciding whether or not to relay a message to a node. However, since the Neural Network variant performs objectively better than the Decision Tree variant, we refer to the Neural Network approach used in MLProph as MLProph from here on. In KNNR, the authors use certain context information as input for the K-Nearest Neighbor classifier to perform efficient routing.

Context-oblivious routing protocols consist of protocols such as Spray and Wait [13], and Epidemic routing [14]. Epidemic routing performs message *flooding* to increase the probability of message delivery. While Epidemic routing is simple, it increases the overhead on the network and does not yield good results. Spray and Wait routing improves on Epidemic routing, and performs controlled flooding. The algorithm decides the number of copies to be flooded (denoted as L) as well as which nodes to relay these copies to.

Recently, research in OppNets has also started focusing on tackling security issues [15], design of trust based mechanisms for counteracting selfish node behavior [16], [17] and energy efficiency [18].

III. Motivation

As mentioned in Section I, we believe that a study of the data that a ML classifier is to be trained on, forms the basis for achieving optimal performance in classification. In this paper, we build upon the core ideas of the MLProph protocol, and thus analyze the data used by the authors in that paper. The authors state that they generate data using the Epidemic routing protocol. We perform simulations on the Opportunistic Network Environment (ONE) simulator [19] to generate data with the simulation settings defined in Table III, Section V using Epidemic Routing. The results obtained are shown in Table I. We observe that out of 1466 messages created, only 565 were delivered, resulting in a delivery probability of 38.54%. Therefore, since MLProph uses a binary classification between *message delivered* and *message not delivered* to train on, a class imbalance problem exists. We seek to solve this problem by using cascade learning as mentioned in Section I. We propose a two-stage classifier consisting of a Logistic Regression classifier as the first stage and the same Neural Network employed by MLProph in the second stage.

The proposed ML cascade works as follows: the input data used for MLProph is first provided as input to the Logistic Regression classifier which then generates two probabilities $P_{msg\_delivered}$ and $P_{msg\_not\_delivered}$ for each sample present in the data. These two *features* of obtained probability data are now appended to the original dataset and fed as input to the Neural Network classifier. Thus, a knowledge transfer has taken place between the cascades resulting in improved classification. We opt for just a two-stage cascade as well as Logistic Regression at the input because of the following reasons:

- Using a large number of cascades of ML classifiers adds significant overhead and processing time to the algorithm. Therefore, we simply opt for a two stage classifier to improve the routing decisions over regular MLProph.
- Moreover, we opt for Logistic Regression because of it's inherent simplicity and low computation complexity as compared to classification algorithms such as Random Forests, Neural Networks and Support Vector Machines. This also ensures that the proposed cascaded approach is lightweight, while not compromising on classification capability.

To empirically adjudge whether Logistic Regression is the best choice for the input cascade, we find the time it takes to train the chosen input cascade classifier on the original dataset. We present the build times for training a number of input cascade classifiers using the Weka Java ML library, in Table II. It can be clearly seen that Logistic Regression is the optimal choice for our approach.

TABLE I: Epidemic Routing data analysis

Messages Created	1466
Messages Delivered	565
Delivery Probability	**38.54%**
Messages Started	752654
Messages Relayed	723573
Messages Dropped	721547

TABLE II: Build times for input cascade classifier choices

ML algorithm	Build time (seconds)
Logistic Regression	**1.372254129**
Random Forest	2.971062915
Support Vector Machine	1.988392350
MLP Neural Network	6.921582045

IV. Proposed Solution

In the last section, we justified our motivation for improving routing in OppNets through ML cascades, and gave a brief overview of the proposed protocol. Moreover, from here on, we will refer to our approach as the CAML (CAscaded Machine Learning based) routing protocol. The CAML protocol is built around the core ideas espoused by MLProph, which involve first extracting certain input features from the nodes, and then using these as input to a pre-trained classifier (an MLP Neural Network) to generate a possible value of delivery probability from the learned distribution. This is referred to as the ML probability (P_m). If P_m is greater than $K * P_r$ (where K is a normalization factor and P_r is the estimated delivery probability computed by the PRoPHET protocol), the current sender node at hand will choose to relay the message to the receiver node. Also, in the MLProph paper no explicit value for K has been mentioned, but we achieved good performance for $K = 0.8$.

In CAML based routing, the only difference is an addition of the Logistic Regression classifier in the calculation of P_m. The Logistic Regression classifier is fed the input data, and computes two probabilities for the two classes– *message delivered* ($P_{msg\_delivered}$) and *message not delivered* ($P_{msg\_not\_delivered}$). These are then appended to the original dataset as features. This new dataset is then fed to a MLP classifier configured similarly as in MLProph to generate $P_{m(CAML)}$ from the learned probability distribution for the *message delivered* class.

To summarize, the CAML routing protocol consists of the following main steps:

1) Training phase and building classifiers
2) Computing P_r
3) Predicting $P_{m(CAML)}$ from cascaded classifiers

These steps are detailed in the following subsections:

A. Training phase and building classifiers

Similar to MLProph, we first generate data by employing Epidemic routing. The data generated consists of similar features as those used by MLProph:

- PRoPHET probability P_r
- The buffer capacity of the receiver node
- The number of successful message deliveries for the sender-receiver node pair
- The ratio of successful deliveries to total transfers initiated by the sender-receiver node pair
- The receiver node's speed in m/s
- The sender node's energy level in Joules
- The receiver node's energy level in Joules
- The time elapsed since the message was created, in seconds
- The distance of the relay from the destination of the message in meters
- The message's current hop count (total number of nodes it has traveled to so far)

Here, the training phase is slightly different because multiple classifiers need to be trained in a sequential manner. The first step involves training the Logistic Regression classifier on the input data.

1) Training the Logistic Regression classifier: The Logistic Regression binary classifier [6] is a generalized linear model that aims to minimize the following cost function given n samples in data matrix X and the target labels/classes as a vector y:

$$J(w) = -\frac{1}{n}(\sum_{i=1}^{n}(y_i.\log(h_w(X_i))+(1-y_i).\log(1-h_w(X_i)))) \quad (1)$$

$$h_w(x) = \frac{1}{1+e^{-w^T x}} \quad (2)$$

Thus, the input data is fed to the classifier which aims to minimize the above cost function by *finding* the ideal values of the weights vector w to do so, using an algorithm such as gradient descent. Our two classes, *message delivered* and *message not delivered* are represented as 1 and 0 in equation 1, and populate the target vector y accordingly. Since $h_w(x)$ in equation 2, is the sigmoid function, it actually represents the predicted probability for message transmission, $P_{msg\_delivered}$. Once the optimal values of the weights are obtained we find $P_{msg\_delivered}$ and $P_{msg\_not\_delivered}$ $(1 - P_{msg\_delivered})$ and append these to our original data matrix X, to form X'. We use X' to train the MLP classifier in the second stage of the cascade.

2) Training the MLP classifier at the second stage: The MLP classifier trains on the X' dataset engineered in the previous step. Each input sample from X' is used to predict the classes (message delivered or not delivered) at the output layer of the neural network by assigning a probability to each class label. The Neural Network model is a simple Multilayer Perceptron model, where we have one hidden layer, an input layer and an output layer. The input layer possesses m neurons or nodes, where m is the number of features for our input data. It is important to note, that the current number of features are 12 (that is, $m = 12$), and the previous Logistic Regression classifier had 10 (or $m-2$) features. The output layer has just a single neuron since we are dealing with a binary classification problem.

For the hidden layer, we set the number of neurons to the mean of the number of neurons in both the input and the output layers. This is done as setting a value between the number of

nodes in the input and the number of nodes in the output layers is generally empirically robust [20]. Thus, the number of neurons in the hidden layer is set to be 7 (integer value rounded off from $[\frac{m+1}{2}] = 7$).

The MLP model is a feedforward Neural Network, which means that the inputs from the input layer are multiplied with all the weights assigned to each of the neurons in the hidden layer. At each neuron of the hidden layer the vector product is then passed through a nonlinear function known as the *activation* function. This activation function can be chosen to be the sigmoid function ($f_a(x) = \frac{1}{1+e^{-x}}$) or the hyperbolic tangent function ($f_a(x) = \tanh(x)$). This final set of obtained values is then summed and multiplied with the weight of the output neuron and then passed through the activation function to generate a predicted probability value (as sigmoid outputs value between 0 and 1). Mathematically, this amounts to the following value obtained at the output layer y' (considering n input samples in X'):

$$y' = f_a(\sum_{i=1}^{[\frac{m+1}{2}]} w_{i_{(output)}}.H_i) \tag{3}$$

$$H_i = f_a(\sum_{j=1}^{n} w_{ji}.X'_j) \tag{4}$$

$$f_a(x) = \frac{1}{1+e^{-x}} \tag{5}$$

However, this explains how data flows in the MLP and how the Neural Network architecture computes the required probability $P_{m(CAML)} = y'$ (output classes' representation for message delivered class and not delivered class are still 0 and 1, respectively, as at the input cascade) at the output node after training has been completed and weights have been assigned. However, this does not entail how the ideal choice of weights are discovered during training. For that purpose, we use the backpropagation algorithm. The backpropagation algorithm trains the MLP by aiming to minimize an error function. This error function computes the square of the difference between the value predicted and the actual target value. If the predicted class is a vector y' for n input samples and the actual target is the vector y, the error function is defined as follows:

$$E(w) = \frac{1}{2n}\sum_{i=1}^{n}(||y'_i - y_i||^2) \tag{6}$$

The weight updation rule through backpropagation, is defined as:

$$\Delta w = -\alpha\frac{\partial E}{\partial w} \tag{7}$$

where α represents a term known as the *learning rate*. In this way we can obtain the new updated weight value by adding the previous weight value to the calculated change: $w_{new} = \Delta w + w_{old}$. The weight updation process is undertaken till the error function becomes an acceptably small value. α is generally chosen through experimentations but too small a value can result in increased computation times toward finding the minimum possible value for the error function, and too high a value can lead to possible divergence from the actual solution. Moreover, training for the MLP can be performed one sample at a time (*on-line learning*), or for all the training data collectively (*batch learning*). We opt for batch learning because in on-line learning the weight updations can be chaotic and true backpropagation is not performed.

B. Computing P_r

P_r is computed for a sender-receiver node pair in the PRoPHET protocol [2] using the following set of equations:

$$P_r = P_{r_{(old)}} + (1 - P_{r_{(old)}}) * P_{init} \tag{8}$$

In the above equation, P_{init} is an initialization constant. Moreover, equation 8 shows that nodes that have a high number of interactions with each other will be assigned high probabilities of future interactions as well. However, for nodes that do not meet frequently, an aging process is undertaken:

$$P_r = P_{r_{(old)}} * (\gamma)^k \tag{9}$$

Here, k is the time elapsed since the aging process was previously invoked and γ is the aging factor. Also, PRoPHET assumes a transitive nature of the nodes so if node a is frequently interacting with node b and node b is frequently interacting with node c, then even message transmission from node a to node c would be imputed a high delivery probability P_r through the following equation:

$$P_r = P_{r_{(old)}} + (1 - P_{r_{(old)}}) * P_{r(a,b)} * P_{r(b,c)} * \beta \tag{10}$$

Here, β is a constant named as the transitivity scaling constant.

C. Predicting $P_{m(CAML)}$ from cascaded classifiers

Now that we have the trained cascaded ML model and the PRoPHET probability P_r, we can make the decision to relay the message or not. For each sender-receiver pair, where the sender wishes to relay a message, we first extract all the features from the nodes as in the training phase. Then this input vector is fed as input to the first stage of the cascade, the Logistic Regression classifier. As in the training phase, probabilities $P_{msg\_delivered}$ and $P_{msg\_not\_delivered}$ are appended to the input instance to the first stage and then provided as input to the second stage of the classifier. This classifier is the MLP classifier and generates a prediction probability ($P_{m(CAML)}$) (using the feedforward mechanism illustrated in the training phase) for the *message delivered* class to decide if the message should be relayed or not. This is done using the following rule: if $P_{m(CAML)} > K * P_r$, the message is relayed, otherwise the sender node waits to come in contact with another node more suitable for routing. Then the entire process (excluding the training phase) is repeated again. The initial training phase of the CAML routing protocol is detailed in Algorithm 1 and the general CAML algorithm for routing is detailed in Algorithm 2. For simplicity in demonstration we use on-line learning for training in Algorithm 1.

Algorithm 1 CAML (Training Phase)

Input: matrix X
Output: Trained cascade of classifiers C_1 and C_2
$X' = X$
for each sample index i in dataset X **do**
    train Logistic Regression classifier C_1 on $X(i)$
    obtain $P_{msg\_delivered}$, $P_{msg\_not\_delivered}$ from C_1
    append $P_{msg\_delivered}$, $P_{msg\_not\_delivered}$ to $X'(i)$
for each sample index i in dataset X' **do**
    train MLP classifier C_2 on $X'(i)$

Algorithm 2 CAML Routing Protocol

for each encounter between sender-receiver node pair (n_s, n_r) **do**
    extract feature set $f_{(s,r)}$ from (n_s, n_r)
    obtain $P_{msg\_delivered}$, $P_{msg\_not\_delivered}$ from $C_1(f_{(s,r)})$
    append $P_{msg\_delivered}$, $P_{msg\_not\_delivered}$ to $f_{(s,r)}$
    obtain $P_{m(CAML)}$ from $C_2(f_{(s,r)})$
    compute P_r
    if $P_{m(CAML)} > (K * P_r)$ **then**
        relay message
    else
        continue

V. Results and Analysis

To obtain the results for CAML routing, we simulate and compare it with the HBPR, PRoPHET, MLProph and KNNR protocols. We obtain results across a wide variety of performance metrics using the Java based Opportunistic Network Environment simulator. The settings used in the simulations are detailed in Table III. We compare performance across four different different metrics/parameters:

- Message Delivery Probability: Signifies the number of messages which were successfully delivered to the destinations. The higher the delivery probability, the better the routing protocol.
- Average Hop Count: Signifies the average number of relay nodes through which the message *hopped* or passed to, before arriving at the destination. The lower the hop count, the better the routing protocol.
- Packets Dropped: Signifies the number of dropped message packets from the buffers of the nodes. The lower the number of packets dropped, the better the routing protocol.
- Network Overhead Ratio: Signifies the average number of forwarded copies for each message, thus representing the overall overhead experienced by the network. The lower the network overhead ratio, the better the routing protocol.

We obtain results first using the default settings of Table III, and then by varying certain setting parameters. The results subsequently shown demonstrate that CAML outperforms all the other routing algorithms by a large margin, and across all metrics. The parameters we vary are as follows:

TABLE III: Default simulation settings

Simulation Parameter	Value
Simulation Time	43200 s
Common Interface	Bluetooth (BT)
BT Transmit Speed	500 kBps
BT Transmit Range	100 m
Tram Interface	High-speed (HS)
HS Transmit Speed	10 MBps
HS Transmit Range	1500 m
Host Groups	6
Movement Model	ShortestPathMapBasedMovement
Host Buffer Size	15 MB
Message TTL	300 min
Host Initial Energy	18 kJ
Host Scan Energy	0.4 J
Host Transmit Energy	0.5 J
Host Scan Response Energy	0.4 J
Host Base Energy	0.04 J
Pedestrian Group(s)	2
Car Group(s)	1
Tram Group(s)	3
Pedestrian Speed	4.5-6.5 m/s
Pedestrian Wait Time	0-120 s
Car Speed	2.7-13.9 m/s
Tram Speed	7-10 m/s
Tram Wait Time	10-30 s
Tram Movement Model	MapRouteMovement
Message Generation Interval	25-35 s
Message Size	250-750 kB

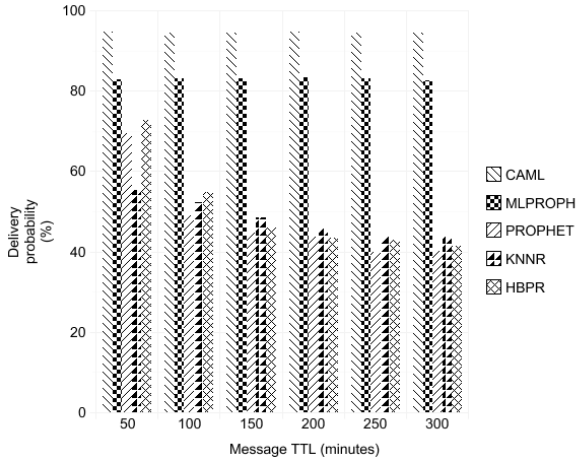

Fig. 1: Message TTL vs Delivery probability

- Message Time-to-Live (TTL): The TTL of the messages are varied as: 50 minutes, 100 minutes, 150 minutes, 200 minutes, 250 minutes, 300 minutes.
- Simulation Time: The simulation time is varied as: 10800 seconds (3 hours), 21600 seconds (6 hours), 32400 seconds (9 hours), 43200 seconds (12 hours).
- Message Generation Interval: The message generation interval signifies how frequently a new message is created and is varied as follows: 5-15 seconds, 15-25 seconds, 25-35 seconds, 35-45 seconds, 45-55 seconds.

A. Results on varying Message TTL

Figure 1 shows the results for delivery probability for the protocols while the message TTL is varied. It can be

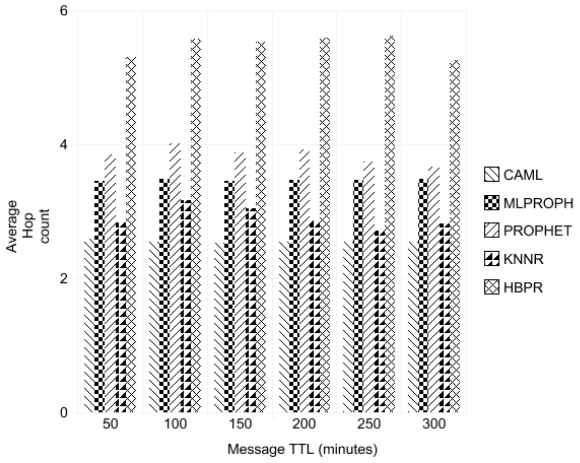

Fig. 2: Message TTL vs Average hop count

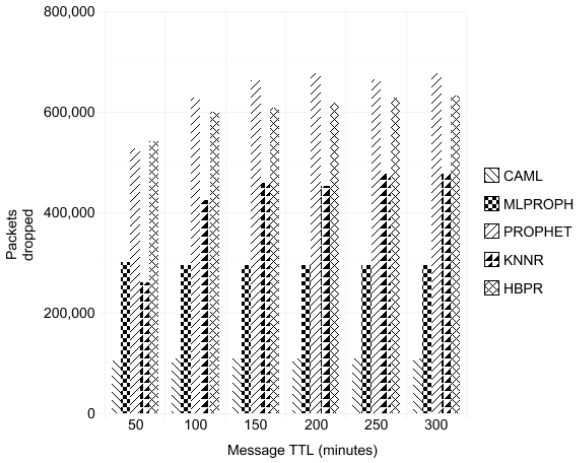

Fig. 3: Message TTL vs Number of packets dropped

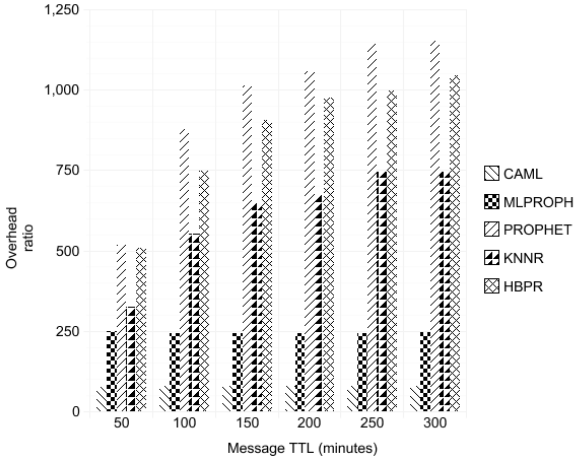

Fig. 4: Message TTL vs Network overhead ratio

seen that CAML outperforms all the other protocols here, as it's delivery probability with respect to the values from the TTL of 50-300 minutes change as 94.75%, 94.54 %, 94.47%, 94.61%, 94.47%, 94.34%. These values are all greater than even the second best performer, MLProph which varies as 82.74%, 83.02%, 83.08%, 83.22%, 83.15%, 82.67%. The other algorithms perform far poorly, with minimum-maximum values ranging between 43.72% (TTL 300) to 55.25% (TTL 50) for KNNR, 39.84% (TTL 250) to 69.44% (TTL 50) for PRoPHET, and 41.34% (TTL 300) to 72.71% (TTL 50) for HBPR. The trend also shows that increasing TTL leads to a drop in the delivery probability for the routing protocols, on average. Moreover, CAML performs the most optimally, with the highest possible value of delivery probability reaching 94.75% for a TTL of 50 minutes.

CAML's superlative performance can also be seen in Figure 2, which measures the values of average hop count, as the TTL is varied from 50-300 minutes. CAML has the lowest values of average hop count across all the routing protocols which are: 2.5947, 2.5512, 2.5336, 2.5479, 2.5516, 2.5466. The second best performer is KNNR, with average hop count values: 2.8358, 3.1654, 3.0381, 2.8739, 2.7165, 2.8222. The other protocols do not compare well and even have minimum values for average hop count greater than 3. CAML again performs optimally in this scenario and results in the lowest average hop count of 2.5336 for a TTL of 150 minutes.

In terms of the packets dropped, CAML is again the top performing routing protocol as can be observed from Figure 3. The results for CAML for the number of packets dropped while the TTL is similarly varied from 50-300 minutes are: 105839, 108470, 108519, 108765, 109136, 108473. MLProph is the second best, but even then performs poorly relative to CAML. The minimum and maximum number of packets dropped for MLProph are 295418 (TTL 250) and 301001 (TTL 50), respectively. These are still worse from CAML by at least a factor of 2.5.

For the network overhead ratio performance metric as well, CAML shows the best performance with the lowest values of overhead ratio for all the varied TTL values. In Figure 4, it can be seen that the network overhead ratio for CAML remains relatively constant, with values only slightly increasing as TTL is varied from 50 to 300 minutes: 77.0720, 79.2446, 79.3740, 79.4311, 79.8383, 79.4497. The other algorithms do not compare well here as well, and have much higher values for the overhead ratio which are greater to CAML by at least a factor of 3. MLProph is the next best performing protocol but even the minimum value exhibited by it is 243.3574 (TTL 200) which is much higher than the minimum value exhibited by CAML, 77.0720 (TTL 50).

Thus it can be inferred that due to the addition of classifier cascades, CAML performs much more optimally than the other protocols. The robust performance is attributed to the use of the Logistic Regression classifier in a cascade with the MLP Neural Network.

B. Results on varying Simulation Time

The results obtained for the performance metrics while simulation time is varied are depicted in Figures 5-8. In

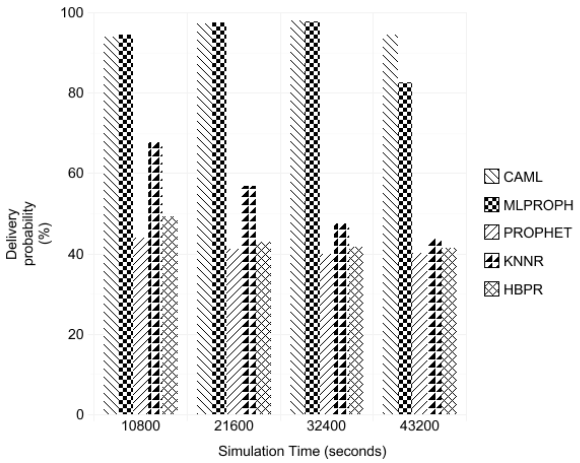

Fig. 5: Simulation time vs Delivery probability

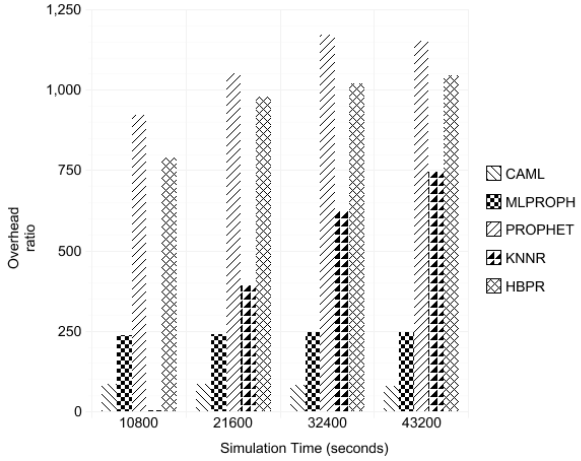

Fig. 8: Simulation time vs Network overhead ratio

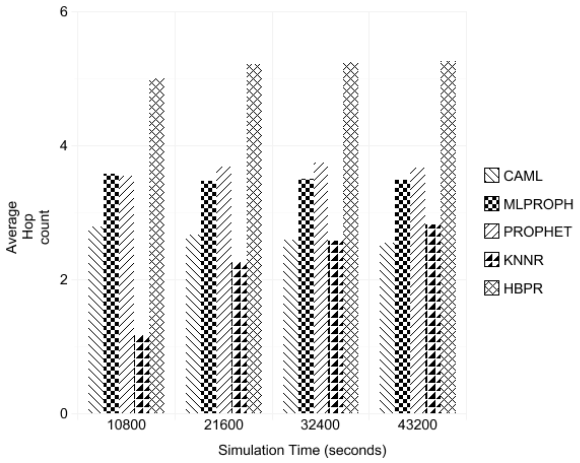

Fig. 6: Simulation time vs Average hop count

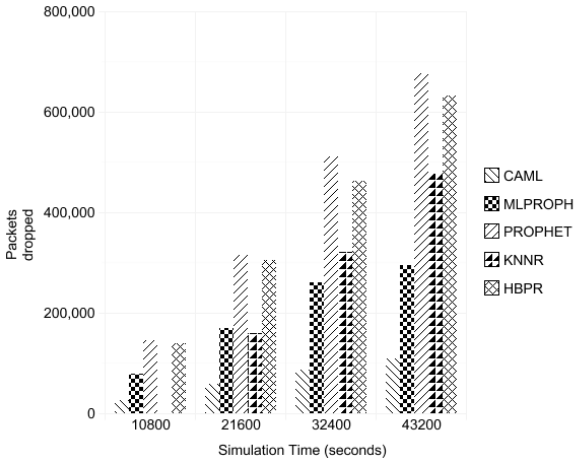

Fig. 7: Simulation time vs Number of packets dropped

Figure 5, which shows the variation of delivery probability with simulation time, MLProph and CAML are the top performers. It is only here that MLProph shows comparative and even minutely better performance (for simulation time values: 10800 seconds and 21600 seconds) than CAML. We attribute this fact to the idea that CAML is a more robust ML model, it makes more intelligent, efficient and optimal routing decisions that do not pay off in the short-term but in the long-term. To exemplify, if we look at Figure 5, we can see that the delivery probability values for CAML as simulation time is varied from 10800-43200 seconds are: 93.97%, 97.26%, 97.90%, 94.34%. Whereas for MLProph, these values are 94.52%, 97.40%, 97.81%, 82.67%. This data shows us that CAML makes some slightly different routing decisions for messages which lead to minutely lower performance initially, that is for the 10800 and 21600 seconds simulation time values, but pay off over the long-run. As the simulation time is increased, we see that these routing decisions were vital to continued top-tier performance as CAML outperforms MLProph in both the 32400 and 43200 seconds simulation time values. Eventually for a reasonable 12 hour simulation (43200 seconds) MLProph's delivery probability value drops to 82.67%, while CAML obtains 94.34% for the same.

Similarly in Figure 6, where we measure the average hop count with respect to the simulation time, we see that initially CAML makes routing decisions that pay off over the long-term. KNNR and CAML can be seen as the top performers. Also, it is important to note that we disregard KNNR's average hop count value for the simulation time of 10800 seconds. This is because the core ML algorithm has not been invoked up to that point and is still collecting values for training/classification, making CAML the optimal choice. However, for 21600 seconds of simulation, KNNR performs better than CAML, as CAML has a value of 2.6606 compared to KNNR's 2.2512. Although, KNNR's short-term better performance is trumped by CAML's long-term decision-making as for 43200 seconds of simulation time, CAML obtains an average hop count of 2.5466 compared to KNNR's average hop count of 2.8222. Thus, it can be seen that CAML shows

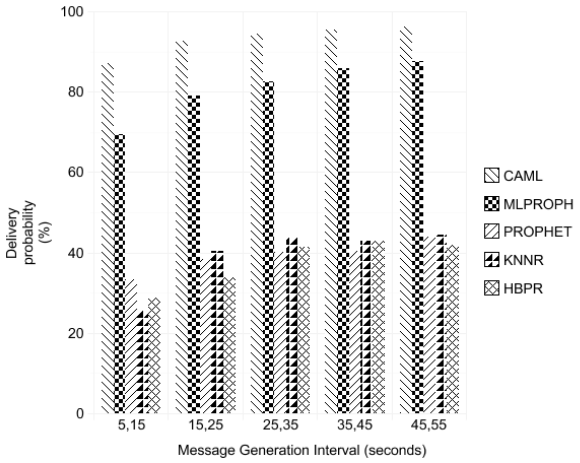

Fig. 9: Message Generation Interval vs Delivery probability

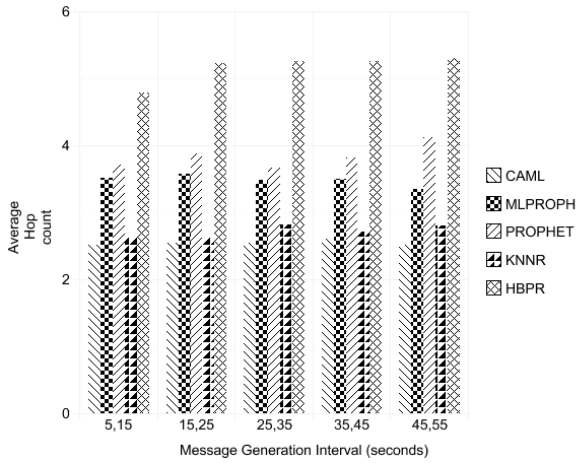

Fig. 10: Message Generation Interval vs Average hop count

a better sustained performance all throughout the simulation.

In Figure 7, we see the variation in packets dropped as simulation time is increased from 10800-43200 seconds. Here again, we disregard the simulation of 10800 seconds for KNNR due to the reason mentioned above. It can be seen that CAML is the top performer and has the lowest number of packets dropped as simulation time is increased with values: 27116, 58477, 85879, 108473. MLProph obtains the following results: 78954, 169203, 260516, 295762. Finally, KNNR obtains the following results: N/A (reason listed in previous paragraph), 159921, 321929, 476086. Thus, it can be seen that CAML is able to maintain optimal performance throughout while MLProph and KNNR compare relatively poorly. Also, KNNR performs better than MLProph only for 10800 seconds of simulation time and in general, performs poorly. PRoPHET and HBPR also do not perform well compared to the other protocols.

Figure 8 shows the variation of network overhead ratio with increasing simulation times from 10800-43200 seconds. Here again, CAML performs optimally compared to the other algorithms and obtains the following values: 86.4198, 85.3775, 81.8358, 79.4497. After CAML, the second best performer is MLProph which obtains the following results: 236.3101, 241.0872, 244.9879, 245.2434. The other algorithms do not compare favorably. Thus, it can be seen that CAML's overhead ratio results are much smaller than even MLProph's, leading to much lower resource overhead on the network as a whole when CAML is employed.

CAML is able to maintain sustained optimal performance throughout the simulations compared to the other protocols.

C. Results on varying Message Generation Interval

Next, we vary the message generation interval from a new message appearing between every 5-15 seconds, 15-25 seconds, 25-35 seconds, 35-45 seconds and 45-55 seconds. Figure 9 shows the results obtained for the delivery probability while varying the message creation interval, as above. Here the performance of CAML is much higher for the performance metric compared to the other routing protocols with values obtained as 87.13%, 92.80%, 94.34%, 95.42%, 96.10%. MLProph is the next best protocol in terms of performance but relative to CAML, it is still not optimal. The other routing protocols do not fare well and get the delivery probability values only slightly above the 40% mark. Moreover, the trend shows that increasing the message generation interval leads to an increase in delivery probability, which is accounted for by the fact that more messages are being generated rapidly in the same time instants, leading to more deliveries. This in turn increases the probability of delivery.

Figure 10 shows the change in the average hop count values as the message generation interval is varied. Here CAML again outperforms the other routing protocols. The values obtained for CAML are: 2.5141, 2.5473, 2.5466, 2.6033, 2.5173. KNNR is the second best choice with values obtained being: 2.6428, 2.6235, 2.8222, 2.7143, 2.8041. The rest of the protocols have much higher values of average hop count. Thus, CAML manages to achieve the lowest possible average hop count value for message transmission, with a hop count of 2.5141 (message generation interval: 5-15 seconds) and maintains optimal performance throughout.

Figure 11 depicts the number of packets dropped as the message generation interval is varied. CAML is the top performer and outperforms the other routing protocols by quite a large margin. The results obtained for CAML are: 171942, 130225, 108473, 95754, 82354. MLProph is the second best performer but is still under-performing relative to CAML. The values obtained by MLProph are: 390697, 338339, 295762, 259642, 226710. Thus it is evident that MLProph still leads to more dropped packets relative to CAML. Moreover, Figure 11 shows that the general trend is that the number of packets dropped decrease as the message generation interval increases. This can be imputed to the fact that since more messages are being delivered (as mentioned in the previous paragraph), fewer of them are being dropped leading to improvements.

Finally, Figure 12 depicts the network overhead ratio as the message generation interval is increased. Here too, CAML has the lowest values compared to all the other routing protocols: 43.0081, 63.5046, 79.4497, 93.7416, 101.0143. Amongst the remaining protocols, MLProph shows the next

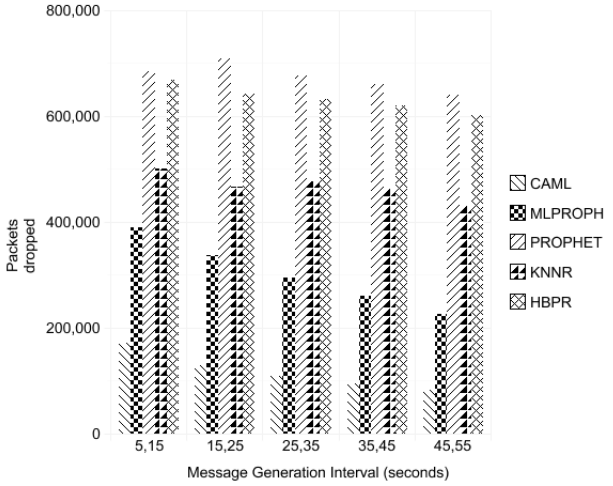

Fig. 11: Message Generation Interval vs Number of packets dropped

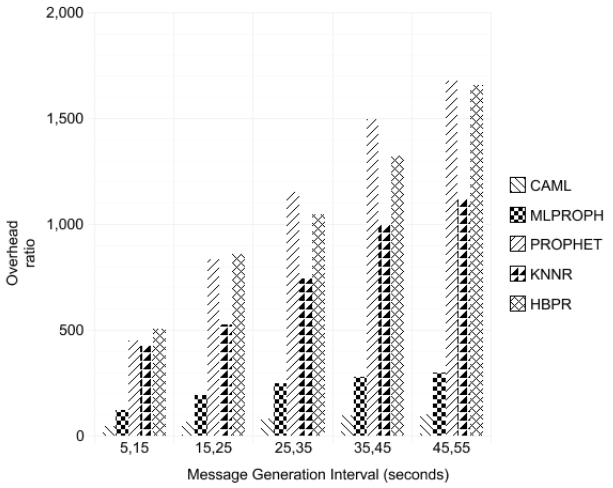

Fig. 12: Message Generation Interval vs Network overhead ratio

best performance with relatively large values compared to CAML: 123.0427, 192.9431, 245.2434, 279.0277, 300.0721. Moreover, the trend that is observed is that increasing the message generation interval increases the network overhead ratio, which fits with the logical idea that since more messages are being created in the same time steps, the network's overhead and resource utilization will increase.

CAML has been shown to outperform the other algorithms in terms of all the performance criteria while the message generation interval is increased.

VI. Conclusion

In this paper, we have presented an approach to improve routing for OppIoTs using a cascade learning based ML approach for routing, called CAML. We improve upon the ideas presented in the recent MLProph routing protocol and show that by adding a simple Logistic Regression classifier as an input cascade to an MLP Neural Network classifier, performance can increase manifold. We also perform extensive simulations using the ONE simulator to empirically display the superlative performance exhibited by CAML, compared to other ML and non-ML based routing protocols in status quo.

References

[1] B. Guo, D. Zhang, Z. Wang, Z. Yu, and X. Zhou, "Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things," *Journal of Network and Computer Applications*, vol. 36, pp. 1531–1539, 2013.

[2] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7(3), pp. 19–20, July 2003.

[3] S. K. Dhurandher, D. K. Sharma, I. Woungang, and S. Bhati, "Hbpr: History based prediction for routing in infrastructure-less opportunistic networks," in *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 931–936, March 2013.

[4] D. K. Sharma, S. K. Dhurandher, I. Woungang, R. K. Srivastava, A. Mohananey, and J. J. P. C. Rodrigues, "A machine learning-based protocol for efficient routing in opportunistic networks," *IEEE Systems Journal*, pp. 1–7, 2017.

[5] D. K. Sharma, Aayush, A. Sharma, and J. Kumar, "Knnr: k-nearest neighbour classification based routing protocol for opportunistic networks," in *Tenth International Conference on Contemporary Computing (IC3)*, pp. 1–6, Aug 2017.

[6] M. Tranmer and M. Elliot, "Binary logistic regression," *Cathie Marsh for Census and Survey Research, Paper, vol. 20, 2008.*

[7] P. F. Christ et. al, "Automatic liver and lesion segmentation in CT using cascaded fully convolutional neural networks and 3D conditional random fields," *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 415–423, 2016.

[8] S. M. Sarwar, M. Hasan, and D. I. Ignatov, "Two-stage cascaded classifier for purchase prediction," *arXiv preprint arXiv:1508.03856*, 2015.

[9] M. Simonovsky and N. Komodakis, "Onionnet: Sharing features in cascaded deep classifiers," *arXiv preprint arXiv:1608.02728*, 2016.

[10] M. Musolesi and C. Mascolo, "Car: Context-aware adaptive routing for delay-tolerant mobile networks," *IEEE Transactions on Mobile Computing*, vol. 8(2), pp. 246–260, Feb 2009.

[11] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks," In *Proceedings 25TH IEEE INFOCOM International Conference on Computer Communications*, pp. 1–11, April 2006.

[12] C. Boldrini, M. Conti, J. Jacopini, and A. Passarella, "Hibop: a history based routing protocol for opportunistic networks," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1–12, June 2007.

[13] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," in *Proceedings of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, WDTN '05, pp. 252–259, New York, NY, USA, 2005.

[14] A. Vahdat, David Becker, et al, "Epidemic routing for partially connected ad hoc networks," *Technical Report CS-200006*, Duke University, 2000.

[15] A. Chhabra, V. Vashishth, and D. K. Sharma, "A fuzzy logic and game theory based adaptive approach for securing opportunistic networks against black hole attacks," *International Journal of Communication Systems*, vol. 31(4):e3487, 2017.

[16] A. Kumar, et al. "An altruism-based trust-dependent message forwarding protocol for opportunistic networks," *International Journal of Communication Systems*, 30(10):e3232.

[17] A. Chhabra, V. Vashishth, and D. K. Sharma, "Seir: A stackelberg game based approach for energy-aware and incentivized routing in selfish opportunistic networks," in *51st Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, March 2017.

[18] S. T. Kouyoumdjieva and G. Karlsson, "Energy savings in opportunistic networks," in *11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pp. 57–64, April 2014.

[19] A. Keränen, J. Ott, and T Kärkkäinen, "The one simulator for dtn protocol evaluation," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, Simutools, pp. 55:1–55:10, ICST, Brussels, Belgium, 2009.

[20] J. Heaton, "Introduction to neural networks with Java," Heaton Research, Inc., 2008.